INF5620 - Obligatory Exercise 2

Jens Kristoffer Reitan Markussen

October 15, 2013

1 Introduction

Report describing the implementation and numerics used in the second obligatory exercise in the course INF5620 Numerical Methods for Partial Differential Equations.

2 Equations

2.1 Partial Differential Equation

Two-dimensional, linear wave equation, with damping,

$$\frac{\partial^2 u}{\partial t^2} + b\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(q(x,y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(q(x,y)\frac{\partial u}{\partial y}\right) + f(x,y,t)$$

with boundary condition,

$$\frac{\partial u}{\partial n} = 0$$

in a rectangular spatial domain with these conditions

$$\Omega = [0, L_x] \times [0, L_y]$$

$$u(x,y,0) = I(x,y)$$

$$u_t(x,y,0) = V(x,y)$$

2.2 Numerics

PDE in compact finite difference notation

$$[D_t D_t u]^n_{i,j} + b[D_{2t} u]^n_{i,j} = [D_x \bar{q}^x D_x u]^n_{i,j} + [D_y \bar{q}^y D_y u]^n_{i,j}$$

Central difference approximations

$$\frac{\partial^2 u}{\partial t^2} \approx [D_t D_t u]^n_{i,j} = \frac{u^{n+1}_{i,j} - 2u^n_{i,j} + u^{n-1}_{i,j}}{\Delta t^2}$$

$$b\frac{\partial u}{\partial t} \approx b[D_{2t} u]^n_{i,j} = b\frac{u^{n+1}_{i,j} - 2u^{n-1}_{i,j}}{2\Delta t}$$

$$\begin{aligned}\frac{\partial}{\partial x}\left(q(x,y)\frac{\partial u}{\partial x}\right) \approx [D_x \bar{q}^x D_x u]^n_{i,j} &= \frac{1}{\Delta x}\left[q_{i+\frac{1}{2},j}\frac{u^n_{i+1,j} - u^n_{i,j}}{\Delta x} - q_{i-\frac{1}{2},j}\frac{u^n_{i,j} - u^n_{i-1,j}}{\Delta x}\right] \\ &= \frac{1}{\Delta x^2}\left(q_{i+\frac{1}{2},j}(u^n_{i+1,j} - u^n_{i,j}) - q_{i-\frac{1}{2},j}(u^n_{i,j} - u^n_{i-1,j})\right)\end{aligned}$$

$$\begin{aligned}\frac{\partial}{\partial y}\left(q(x,y)\frac{\partial u}{\partial y}\right) \approx [D_y \bar{q}^y D_y u]^n_{i,j} &= \frac{1}{\Delta y}\left[q_{i,j+\frac{1}{2}}\frac{u^n_{i,j+1} - u^n_{i,j}}{\Delta y} - q_{i,j-\frac{1}{2}}\frac{u^n_{i,j} - u^n_{i,j-1}}{\Delta y}\right] \\ &= \frac{1}{\Delta y^2}\left(q_{i,j+\frac{1}{2}}(u^n_{i,j+1} - u^n_{i,j}) - q_{i,j-\frac{1}{2}}(u^n_{i,j} - u^n_{i,j-1})\right)\end{aligned}$$

Relation for creating initial scheme

$$\frac{\partial u}{\partial n} \approx \frac{u^{n+1}_{i,j} - 2u^{n-1}_{i,j}}{2\Delta t}1 = 0$$

$$u^{-1}_{i,j} = u^1_{i,j}$$

Discretization

Make q only valid at the grid points using the arithmetic mean

$$\begin{aligned}q_{i,j+\frac{1}{2}} &= \frac{1}{2}(q_{i,j+1} + q_{i,j}) \\ q_{i,j-\frac{1}{2}} &= \frac{1}{2}(q_{i,j} + q_{i,j-1}) \\ q_{i+\frac{1}{2},j} &= \frac{1}{2}(q_{i+1,j} + q_{i,j}) \\ q_{i-\frac{1}{2},j} &= \frac{1}{2}(q_{i,j} + q_{i-1,j})\end{aligned}$$

Full approximation scheme for $u_{i,j}^{n+1}$

$$u_{i,j}^{n+1} = \left(1 + b\frac{\Delta t}{2}\right)^{-1} \Bigg[2u_{i,j}^{n} + u_{i,j}^{n-1}\left(b\frac{\Delta t}{2} - 1\right) + \frac{\Delta t^2}{2\Delta x^2}((q_{i+1,j} + q_{i,j})(u_{i+1,j}^{n} - u_{i,j}^{n}) - (q_{i,j} + q_{i-1,j})(u_{i,j}^{n} - u_{i-1,j}^{n})) + \frac{\Delta t^2}{2\Delta y^2}((q_{i,j+1} + q_{i,j})(u_{i,j+1}^{n} - u_{i,j}^{n}) - (q_{i,j} + q_{i,j-1})(u_{i,j}^{n} - u_{i,j-1}^{n}))\Bigg]$$

Full approximation scheme at boundary ghost cells $u_{0,j}^{n+1}$, $u_{i,0}^{n+1}$, $u_{L_x,j}^{n+1}$ and u_{i,L_y}^{n+1}, and corner cells $u_{0,0}^{n+1}$, $u_{L_x,0}^{n+1}$,u_{L_x,L_y}^{n+1} and u_{0,L_y}^{n+1}

$$u_{0,j}^{n+1} = \left(1 + b\frac{\Delta t}{2}\right)^{-1} \Bigg[2u_{i,j}^{n} + u_{i,j}^{n-1}\left(b\frac{\Delta t}{2} - 1\right) + \frac{\Delta t^2}{2\Delta x^2}((q_{1,j} + q_{0,j})(u_{1,j}^{n} - u_{0,j}^{n}) - (q_{0,j} + q_{1,j})(u_{0,j}^{n} - u_{1,j}^{n})) + \frac{\Delta t^2}{2\Delta y^2}((q_{0,j+1} + q_{0,j})(u_{0,j+1}^{n} - u_{0,j}^{n}) - (q_{0,j} + q_{0,j-1})(u_{0,j}^{n} - u_{0,j-1}^{n}))\Bigg]$$

Equivalent scheme can be made for the other cells.
When computing u^{n+1} we required knowledge of the mesh points u^n and u^{n-1}, which means that when computing u^1 we require knowledge of u^0 and u^{-1}. u^0 is defined by the initial conditions $I(x, y)$, however we need a modified scheme for u^{-1} as well as $V(x, y)$.

$$u_{i,j}^{-1} = u_{i,j}^{0} + \frac{\Delta t^2}{4\Delta x^2}((q_{i+1,j} + q_{i,j})(u_{i+1,j}^{0} - u_{i,j}^{0}) - (q_{i,j} + q_{i-1,j})(u_{i,j}^{0} - u_{i-1,j}^{0})) + \frac{\Delta t^2}{4\Delta y^2}((q_{i,j+1} + q_{i,j})(u_{i,j+1}^{0} - u_{i,j}^{0}) - (q_{i,j} + q_{i,j-1})(u_{i,j}^{0} - u_{i,j-1}^{0}))$$

Scheme can also be used as initial condition scheme if modified in the same way we modified the inner scheme.

3 Implementation

The implementation is based on handout code `wave2D_u0.py`, so the same base structure was followed when implementing the numerics described above. The program can be executed in two different ways, one with the same initial condition from the handout code, a smooth gaussian function in the center of the domain and the other investigating a physical problem with a sub-sea hill in the middle of the domain.

```
python wave2d_module.py --run_Gaussian

python wave2d_module.py --run_physical_problem
```

The solver is implemented in two ways, regular scalar array iteration, and `numpy` vectorized array computations.

3.1 Verification

Implementation error verification is implemented using the python `nose` unit-test framework. The tests can be ran from the terminal with the command `nosetests -v -s wave2d_module.py`, the runtime option `-v` activates verbose output, and the option `-s` deactivates `stdout` suppression.

Constant Solution

Constant solution verification is implemented by modifying the initial condition to a constant value, $I(x, y) = C$. This should produce a constant solution array.

1D Plug Wave Solution in 2D

1D plug wave solution verification is implemented by modifying the initial condition to be constant in some parts of the domain and zero elsewhere. The 2D program is tested by inserting a 1D plug wave in the x direction with no damping and a constant q.